

















Teoria Grafurilor. Aplicații.

(partea I)

*Alexandru Cohal
Noiembrie 2013*

Cuprins

 Introducere	3
 Reprezentarea Grafurilor în memorie	3
 Reprezentarea prin Matrice de Adiacență.....	3
 Reprezentarea prin Liste de Adiacență.....	4
 Reprezentarea prin Lista Muchiilor / Arcelor.....	5
 Grafuri ponderate	5
 Parcurgerea grafurilor	6
 Parcurgerea în adâncime (DFS).....	6
 Parcurgerea în lățime (BFS).....	7
 Conexitate	8
 Tare - Conexitate	9
 Grafuri hamiltoniene	10
 Grafuri euleriene	11
 Aplicații	12
 Legături	12
 Bibliografie.....	12



Introducere (câteva noțiuni elementare)

Un **graf** (**orientat** sau **neorientat**) este o pereche ordonată de mulțimi
 $G = (V, E)$.

Mulțimea V este o mulțime nevidă și finită de elemente denumite **vârfurile** grafului.

Mulțimea E este o mulțime de perechi de vârfuri din graf. În cazul grafurilor orientate, perechile de vârfuri din mulțimea E sunt ordonate și sunt denumite **arce**. În cazul grafurilor neorientate, perechile de vârfuri din mulțimea E sunt neordonate și sunt denumite **muchii**.

Perechea **ordonată** formată din vârfurile x și y se notează (x, y) ; vârful x se numește **extremitate inițială** a arcului (x, y) , iar vârful y se numește **extremitate finală** a arcului (x, y) .

Perechea **neordonată** formată din vârfurile x și y se notează $[x, y]$; vârfurile x și y se numesc **extremitățile** muchiei $[x, y]$.

Dacă există un arc sau o muchie cu extremitățile x și y , atunci vârfurile x și y sunt **adiacente**; fiecare extremitate a unei muchii / unui arc este considerată **incidentă** cu muchia / arcul respectiv.



Reprezentarea Grafurilor în memorie



Reprezentarea prin Matrice de Adiacență

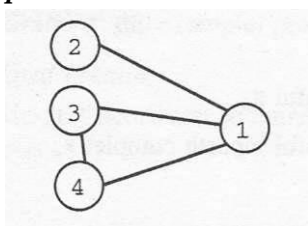
Fie $G = (V, E)$ un **graf neorientat**. Să notăm cu n numărul de vârfuri din graf.

Matricea de adiacență este o matrice pătratică, având n linii și n coloane, cu elemente din mulțimea $\{0, 1\}$, astfel:

$A[i][j] = 1$ dacă există muchia $[i, j]$ în graf

$A[i][j] = 0$ în caz contrar.

Exemplu:



	1	2	3	4
1	0	1	1	1
2	1	0	0	0
3	1	0	0	1
4	1	0	1	0

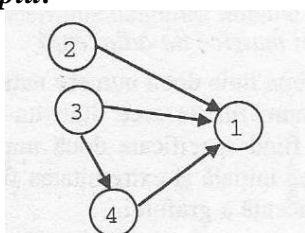
Fie $G = (V, E)$ un **graf orientat**. Să notăm cu n numărul de vârfuri din graf.

Matricea de adiacență este o matrice pătratică, având n linii și n coloane, cu elemente din mulțimea $\{0, 1\}$, astfel:

$A[i][j] = 1$ dacă există arcul (i, j) în graf

$A[i][j] = 0$ în caz contrar.

Exemplu:



	1	2	3	4
1	0	0	0	0
2	1	0	0	0
3	1	0	0	1
4	1	0	0	0

Observații

➤ Matricea de adiacență a unui graf neorientat este simetrică față de diagonala principală, în timp ce matricea de adiacență a unui graf orientat nu este simetrică față de diagonala principală.

➤ Dimensiunea spațiului de memorie necesar pentru reprezentarea unui graf prin matrice de adiacență este $O(n^2)$.

Implementare

➤ Metoda “clasică” – prin declararea și folosirea unei matrici de dimensiunea $n \times n$.

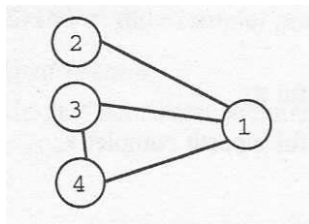


Reprezentarea prin Liste de Adiacență

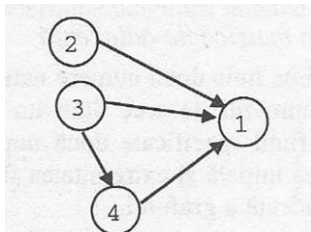
Fie $G = (V, E)$ un **graf neorientat** sau **orientat** cu n vârfuri.

Pentru a reprezenta grafurile prin **liste de adiacență**, vom reține pentru fiecare vârf x al grafului toate vârfurile y cu proprietatea că există muchia $[x, y]$ (pentru graf neorientat), respectiv există arcul (x, y) (pentru graf orientat), formând n liste de adiacență. Ordinea în care sunt memorate vârfurile într-o listă de adiacență nu contează.

Exemple:



	1	2	3	4
1		2	3	4
2	1			
3	1	4		
4	3	1		



	1	2	3	4
1				
2	1			
3	1	4		
4	1			

Implementare

➤ Cu vectori “clasici” – fiecare listă de adiacență este reprezentată ca un vector cu maxim n componente în care vârfurile sunt memorate pe poziții consecutive

➤ Cu liste înlanțuite – fiecare listă de adiacență este reprezentată ca o listă înlanțuită; se reține pentru fiecare element al listei adresa spre următorul element precum și informația utilă

➤ Cu vectori din STL – fiecare listă de adiacență este reprezentată ca un vector din STL (vector).

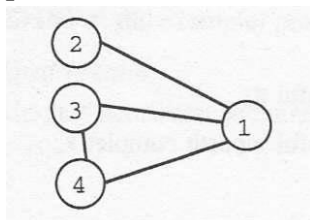


Reprezentarea prin Lista Muchiilor / Arcelor

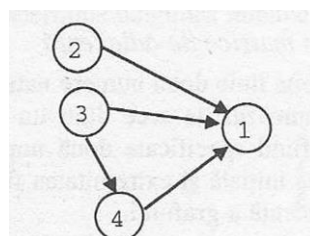
Pentru a reprezenta un **graf neorientat** prin **lista muchiilor**, respectiv un **graf orientat** prin **lista arcelor**, se utilizează un vector cu m componente, unde m este numărul de muchii / arce din graf. Pentru fiecare muchie / arc vor fi reținute **cele două extremități**.

În cazul muchiilor, ordinea extremităților nu contează. În cazul arcelor, va fi reținută mai întâi extremitatea inițială și apoi cea finală.

Exemple:



1	2	3	4	5
1	1	1	3	2
2	3	4	4	3



1	2	3	4	5
2	3	3	4	4
1	1	4	3	1

Implementare

➤ Cu reprezentarea unei muchii / unui arc printr-o **structură** cu două câmpuri; Se va folosi un singur vector de lungime m cu elemente de tip structură; Vectorul poate fi declarat static, dinamic sau ca vector din STL (*vector*)

➤ Prin folosirea a doi vectori de lungime m (sau a unei matrice cu două linii și m coloane); Vectorii / Matricea pot fi definiți static, dinamic sau ca vectori din STL (*vector*).



Grafuri ponderate

În numeroase situații practice modelate cu ajutorul grafurilor, fiecare muchie / arc a / al grafului are asociat un anumit **cost** sau o anumită **pondere** (de exemplu, lungimea cablului necesar pentru conectarea a două calculatoare într-o rețea, costul de transport pe o anumită rută, profitul obținut dintr-o anumită tranzacție etc.).

Graful $G = (V, E)$ (orientat sau neorientat) însoțit de o funcție $c: E \rightarrow R$, prin care se asociază fiecărei muchii / arc din graf un număr real se numește **graf ponderat**.

Pentru a reprezenta un graf ponderat trebuie să reținem și costurile asociate muchiilor / arcelor.

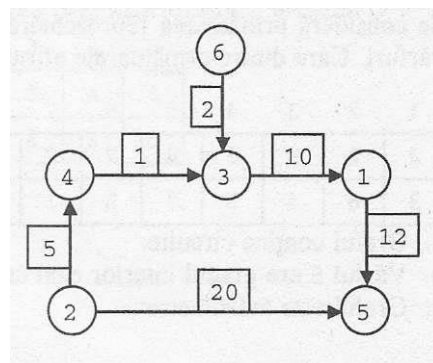
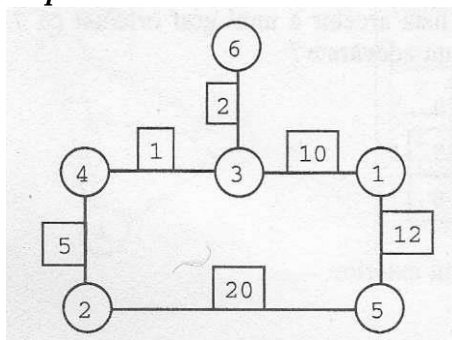
Astfel, **matricea de adiacență** devine matricea costurilor, o matrice pătratică C având n linii și n coloane (unde n este numărul de vârfuri din graf), definită astfel:

$C[x][y]$ va fi costul muchiei / arcului de la x la y dacă există, sau o dacă $x = y$, sau o valoare specială, care depinde de problemă, indicând faptul că nu există muchie / arc de la x la y .

În **listele de adiacență** nu vom reține doar vârfurile adiacente, ci și costurile arcelor / muchiilor corespunzătoare.

În **lista muchiilor / arcelor** vom adăuga pentru fiecare muchie un câmp suplimentar (costul).

Exemple:



Parcurgerea grafurilor

Parcurgerea unui graf presupune examinarea sistematică a vârfurilor grafului, cu scopul prelucrării informațiilor asociate vârfurilor.

Există două metode fundamentale de parcurgere a grafurilor:

- parcurgerea în **adâncime** (*Depth First Search - DFS*)
- parcurgerea în **lățime** (*Breadth First Search - BFS*).



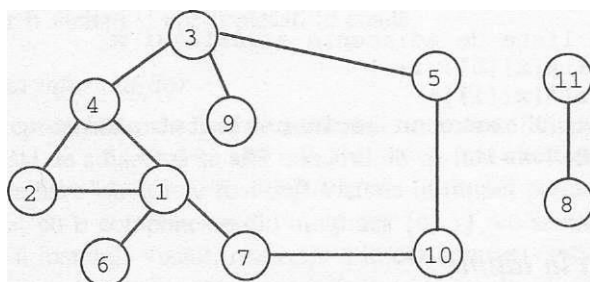
Parcurgerea în adâncime (DFS)

Parcurgerea începe cu un vârf inițial, denumit **vârf de start**. Se vizitează mai întâi vârful de start. La vizitarea unui vârf se efectuează asupra informațiilor asociate vârfului o serie de operații specifice problemei.

Se vizitează apoi primul vecin nevizitat al vârfului de start. Vârful y este considerat vecin al vârfului x dacă există muchia $[x, y]$ (pentru graf neorientat), respectiv arcul (x, y) (pentru graf orientat).

Se vizitează în continuare primul vecin nevizitat al primului vecin al vârfului de start, și așa mai departe, mergând în adâncime până când ajungem într-un vârf care nu mai are vecini nevizitați. Când ajungem într-un astfel de vârf, revenim la vârful său părinte (vârful din care acest nod a fost vizitat). Dacă acest vârf mai are vecini nevizitați, alegem primul vecin nevizitat al său și continuăm parcurgerea în același mod. Dacă nici acest vârf nu mai are vecini nevizitați, revenim în vârful său părinte și continuăm în același mod, până când toate vârfurile accesibile din vârful de start sunt vizitate.

Exemplu:



Parcurgând graful din figură în adâncime considerând drept vârf de start vârful 3 putem obține următoarea ordine de vizitare a vârfurilor accesibile din nodul de start:

3, 4, 1, 2, 6, 7, 10, 5, 9

(Pentru această succesiune, ordinea de vizitare a vecinilor unui vârf este ordinea crescătoare a numerelor lor)

Implementare:

Graful va fi reprezentat prin liste de adiacență (în una dintre cele trei variante). Pentru a reține care vârfuri au fost deja vizitate în timpul parcurgerii vom utiliza un vector *viz*, cu *n* componente din mulțimea {0, 1}, cu semnificația

$viz[i] = 1$ dacă vârful *i* a fost deja vizitat, respectiv 0, în caz contrar.

Observând că ordinea de parcurgere completă a vecinilor unui nod este ordinea inversă a “atingerii” lor, abordarea cea mai simplă folosită pentru parcurgerea efectivă este cea **recursivă**.

Observații:

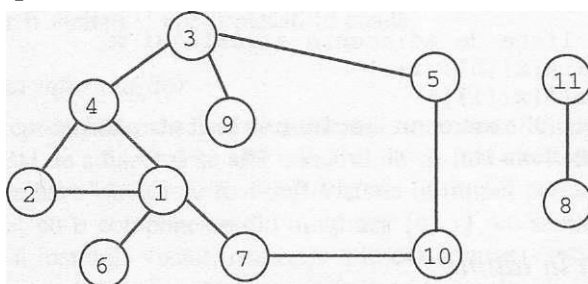
➤ **Complexitatea** parcurgerii în adâncime (DFS) în cazul reprezentării prin liste de adiacență este $O(n + m)$ (în cazul reprezentării prin matrice de adiacență complexitatea este $O(n^2)$).



Parcurgerea în lățime (BFS)

Parcurgerea în lățime începe, de asemenea, cu un vârf inițial, denumit **vârf de start**. Se vizitează mai întâi vârful de start. Se vizitează în ordine toți vecinii nevizitați ai vârfului de start. Apoi se vizitează în ordine toți vecinii nevizitați ai vecinilor vârfului de start și așa mai departe, până la epuizarea tuturor vârfurilor accesibile din vârful de start.

Exemplu:



Considerăm nodul 3 ca nod de start.

Se vizitează mai întâi vârful de start 3. Apoi se vizitează, în ordine, vecinii nevizitați ai lui 3, deci 4, 5 și 9. Se vizitează apoi, în ordine, vecinii nevizitați ai lui 4 (vârfurile 1 și 2), apoi ai lui 5 (vârful 10) și apoi ai lui 9 (care nu are vecini nevizitați). Se vizitează apoi vecinii vârfului 1 (vârfurile 6 și 7) și parcurgerea s-a încheiat (deoarece vârful 2 nu mai are vecini nevizitați, nici vârful 10 și nici vârfurile 6 și 7).

Concluzionând, ordinea în care sunt vizitate vârfurile grafului la parcurgerea BFS cu vârful de start 3 este :

3, 4, 5, 9, 1, 2, 10, 6, 7.

Implementare:

Graful va fi reprezentat prin liste de adiacență (în una dintre cele trei variante). Pentru a reține care vârfuri au fost deja vizitate în timpul parcurgerii vom utiliza un vector *viz*, cu n componente din mulțimea $\{0, 1\}$, cu semnificația

$viz[i] = 1$ dacă vârful i a fost deja vizitat, respectiv 0, în caz contrar.

Observând că ordinea de parcurgere completă a vecinilor unui nod este exact ordinea “atingerii” lor, abordarea cea mai simplă folosită pentru parcurgerea efectivă este cea care folosește o **coadă** (pentru a reține ordinea vizitării elementelor). Această coadă poate fi implementată “clasic” (printr-un vector C cu n elemente; variabilele *prim* și *ultim* rețin poziția de început, respectiv de sfârșit a cozii) sau cu ajutorul STL-ului (*queue*).

Observații:

➤ Parcurgerea în lățime are o proprietate remarcabilă: fiecare vârf este vizitat pe cel mai scurt drum / lanț începând din vârful de start.

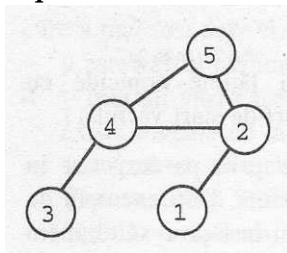
➤ **Complexitatea** parcurgerii în lățime (BFS) în cazul reprezentării prin liste de adiacență este $O(n + m)$ (în cazul reprezentării prin matrice de adiacență complexitatea este $O(n^2)$).



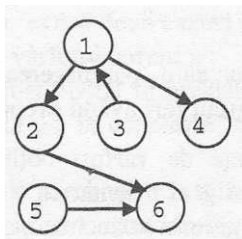
Conexitate

Un graf se numește **conex** dacă oricare ar fi x și y vârfuri din graf există lanț între x și y .

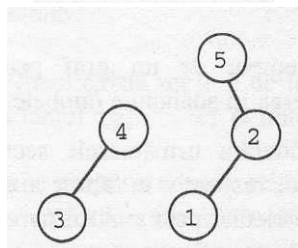
Exemple:



Graf neorientat conex



Graf orientat conex



Graf neorientat neconex (de exemplu, între vârfurile 1 și 3 nu există lanț)

Se numește **componentă conexă** un subgraf conex maximal cu această proprietate (adică, dacă am mai adăuga un vârf și toate muchiile/arcele incidente cu acesta, subgraful obținut nu ar mai fi conex).

Observații:

- Orice graf neconex conține cel puțin două componente conexe.
- Componentele conexe ale unui graf sunt disjuncte.
- Componentele conexe ale grafului constituie o partiție a mulțimii vârfurilor grafului.



Descompunerea unui graf neorientat în componente conexe

A descompune un graf în componente conexe înseamnă a determina toate componentele conexe ale grafului.

A determina componenta conexă a unui vârf x presupune a determina toate vârfurile accesibile din vârfurile x ; deci este suficient să realizăm o parcurgere a grafului (în lățime sau în adâncime) cu vârfurile de start x .

Pentru a descompune grafurile în componente conexe, vom realiza câte o parcurgere pentru fiecare componentă conexă (selectând ca vârf de start vârfurile nevizitate având număr minim).

Observații:

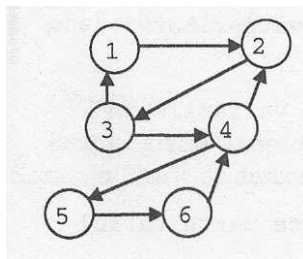
- Pentru a descompune un graf orientat în componente conexe, se va face abstracție de orientarea arcelor.
- Pentru un graf reprezentat prin liste de adiacență, descompunerea în componente conexe utilizând parcurgerea grafului are complexitatea $O(n + m)$. Dacă grafurile sunt reprezentate prin matrice de adiacență, descompunerea în componente conexe utilizând parcurgerea grafului are complexitatea $O(n^2)$.



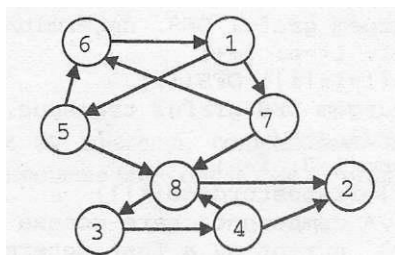
Tare - Conexitate

Un graf orientat se numește **tare-conex** dacă oricare ar fi x și y vârfuri din graf există drum de la x la y și drum de la y la x .

Exemple:



Graf orientat tare-conex



Graf orientat care nu este tare-conex (de exemplu, de la vârfurile 1 la vârfurile 8 există drum, dar de la 8 la 1 nu există)

Se numește **componentă tare-conexă** un subgraf tare-conex maximal cu această proprietate (adică dacă am mai adăuga un vârf și toate arcele incidente cu acesta, subgrafurile obținute nu ar mai fi tare-conexe).

Observație:

- Componentele tare-conexe constituie o partiție a mulțimii vârfurilor grafului.



Descompunerea unui graf orientat în componente tare - conexe

A descompune un graf în componente tare-conexe înseamnă a determina toate componentele tare-conexe ale grafului.

Componentele tare-conexe ale celui de-al doilea graf din exemplul precedent sunt subgrafurile generate de mulțimile de vârfuri: $\{1, 5, 6, 7\}$, $\{3, 4, 8\}$ și $\{2\}$.

Algoritmul Kosaraju-Sharir (1978)

1. Se parcurge graful în adâncime și se numerotează vârfurile grafului în postordine (vârful x este numerotat după ce toți succesorii săi au fost numerotați); în vectorul *postordine* se memorează ordinea vârfurilor.
2. Se determină graful transpus G^T .
3. Se parcurge graful transpus în adâncime, considerând vârfurile în ordinea inversă a vizitării lor în parcurgerea DFS a grafului inițial.
4. Fiecare subgraf obținut în parcurgerea DFS a grafului transpus reprezintă o **componentă tare-conexă** a grafului inițial.

Observație:

- Pentru graful reprezentat prin liste de adiacență, complexitatea algoritmului Kosaraju-Sharir de descompunere în componente tare-conexe este de ordinul $O(n + m)$.



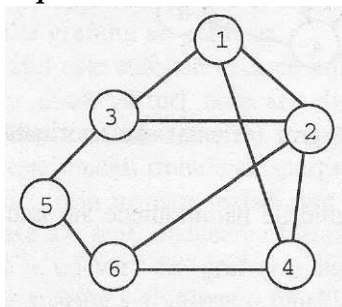
Grafuri hamiltoniene

Un **graf neorientat** se numește **hamiltonian** dacă el conține un ciclu hamiltonian.

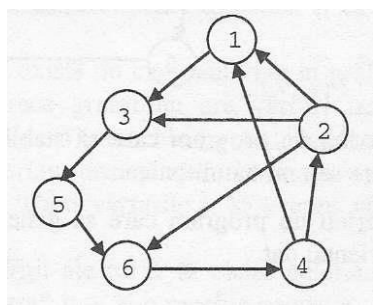
Un **graf orientat** se numește **hamiltonian** dacă el conține un circuit hamiltonian.

Un **ciclu / circuit elementar** se numește **hamiltonian** dacă el trece prin toate vârfurile grafului.

Exemple:



Graf neorientat hamiltonian



Graf orientat hamiltonian



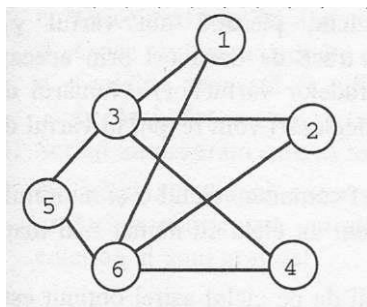
Grafuri euleriene

Un **graf neorientat** se numește **eulerian** dacă el conține un ciclu eulerian.

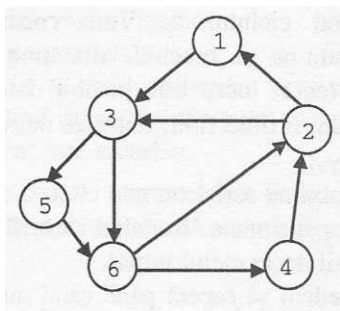
Un **graf orientat** se numește **eulerian** dacă el conține un circuit eulerian.

Un **ciclu / circuit** se numește **eulerian** dacă el trece prin fiecare muchie / arc al grafului exact o dată.

Exemple:



Graf neorientat eulerian (un ciclu eulerian este:
[1, 3, 2, 6, 4, 3, 5, 6, 1])



Graf orientat eulerian (un circuit eulerian este:
[1, 3, 6, 2, 3, 5, 6, 4, 2, 1])



Aplicații

- [Sortaret](#) (Infoarena)
- [Dfs](#) (Infoarena)
- [Bfs](#) (Infoarena)
- [Ctc](#) (Infoarena)

- [Mere](#) (.campion)
- [Prietenii3](#) (.campion)
- [Turn1](#) (.campion)
- [Chei](#) (.campion)
- [Reinvent](#) (.campion)
- [29C](#) (Codeforces)
- [Program1](#) (.campion)
- [Bile1](#) (.campion)
- [Drumuri1](#) (.campion)
- [Coment](#) (.campion)
- [Grafixy](#) (.campion)
- [Sate](#) (Infoarena)
- [Soldati](#) (.campion)
- [Fotbal2](#) (Infoarena)
- [Dfs](#) (.campion)
- [Jungla](#) (.campion)
- [Fazan](#) (.campion)



Legături

- MIT Course: [Graph Teory and Coloring](#)
- MIT Course: [Graph Teory II: Minimum Spanning Trees](#)
- MIT Course: [Graph Teory III](#)



Bibliografie

● Emanuela Cerchez, Marinel Șerban, “Programarea în limbajul C/C++ pentru liceu” volumul III, Editura Polirom, Iași, 2006

Alexandru Cohal
alexandru.cohal@yahoo.com
alexandru.c04@gmail.com
Noiembrie 2013